Модель представления данных при проведении глубокого анализа сетевого трафика*

[1]*А.И. Гетьман <thorin@ispras.ru>*
[1, 2, 3, 4]*В.П. Иванников <ivan@ispras.ru>*
[1]*Ю.В. Маркин <ustas@ispras.ru>*
[1, 2]*В.А. Падарян <vartan@ispras.ru>*
[1]*А.Ю. Тихонов <fireboo@ispras.ru>*
[1] *Институт системного программирования РАН,*
109004, Россия, г. Москва, ул. А. Солженицына, дом 25
[2]*119991 ГСП-1 Москва, Ленинские горы, Московский государственный университет имени М.В. Ломоносова, 2-й учебный корпус, факультет ВМК*
[3]*Московский физико-технический институт (государственный университет),*
141700, Московская область, г. Долгопрудный, Институтский пер., 9
[4]*НИУ Высшая школа экономики,*
Россия, Москва, 101000, ул. Мясницкая, д. 20

Аннотация. В статье предложена объектная модель представления данных при проведении глубокого анализа сетевого трафика. В отличие от модели, используемой большинством существующих сетевых анализаторов, в ней поддерживается восстановление потоков данных, а также проведение их дальнейшего разбора. Тем самым обеспечивается повышение уровня представления (согласно модели OSI) данных, необходимое при анализе сетевого трафика: для понимания механизмов взаимодействия сетевых приложений нужно восстанавливать данные в том виде, в котором этими данными оперируют приложения. На базе предложенной модели реализована инфраструктура для проведения глубокого анализа трафика. Модель предлагает универсальный механизм связывания разборщиков заголовков сетевых протоколов – появляется возможность для независимой разработки функций разбора. Модель также предоставляет функционал для работы с модифицированными (в частности, зашифрованными) данными.

Ключевые слова: анализ сетевого трафика; восстановление потоков данных; модель представления данных; распознавание данных.

DOI: 10.15514/ISPRAS-2015-27(4)-1

* Работа поддержана грантом РФФИ 15-07-07652 А

Для цитирования: Гетьман А. И., Иванников В.П., Маркин Ю.В., Падарян В.А., Тихонов А.Ю. Модель представления данных при проведении глубокого анализа сетевого трафика. Труды ИСП РАН, том 27, вып. 4, 2015 г., стр. 5-22. DOI: 10.15514/ISPRAS-2015-27(4)-1.

1. Введение

В настоящее время задача анализа сетевого трафика приобретает все большую актуальность: этому способствует как развитие и внедрение новых сетевых технологий, так и появление большого количества новых сетевых протоколов прикладного уровня. Перечислим некоторые практические задачи анализа:

- выявление проблем в работе сети
- тестирование (отладка) сетевых протоколов [1, 2]
- сбор статистики, мониторинг сетевых каналов
- предотвращение сетевых атак [3]

Существует большое количество как коммерческих, так и свободно распространяемых сетевых анализаторов [4]. Как правило, каждый инструмент направлен на решение только одной практической задачи, опираясь на "базовый" функционал – разбор заголовков сетевых протоколов и восстановление потоков передаваемых данных.

Большинство существующих сетевых анализаторов поддерживают два режима работы:

- анализ трафика, поступающего на сетевой интерфейс в режиме реального времени (далее *онлайн*-анализ)
- анализ предварительно сохраненных сетевых трасс (далее *оффлайн*-анализ)

В режиме онлайн-анализа инструмент должен работать непрерывно с производительностью, достаточной для разбора трафика, поступающего на сетевой интерфейс. При этом должна обеспечиваться возможность обработки потенциально бесконечного входного потока данных.

В случае оффлайн-анализа инструмент получает входные данные (конечного размера) из файла. Поэтому может проводиться более детальный анализ по сравнению с онлайн-анализом на аналогичном трафике.

На практике в большинстве существующих инструментов оффлайн режим полностью повторяет работу в онлайне за одним исключением: вместо сетевого интерфейса пакеты считываются из файла с сетевой трассой. Отсутствие требований к скорости обработки данных в оффлайн режиме открывает дополнительные возможности:

- визуализировать структуру всех разобранных данных
- анализировать восстановленные потоки данных прикладного уровня
- применять другие разборщики (по сравнению с разборщиком, выбранным инструментом) к данным при просмотре результатов
- проводить отладку модулей разбора заголовков протоколов
- расследовать инциденты нарушения сетевой безопасности

Ограничения на проведение детального оффлайн-анализа главным образом обусловлены архитектурными особенностями инструментов: большая часть анализаторов изначально ориентирована на работу в режиме онлайн. При этом отсутствует возможность дальнейшего разбора восстановленных потоков передаваемых данных. Предлагается реализовать две *независимые* системы: одну для онлайн-, другую – для оффлайн-анализа. Обе системы должны использовать единую инфраструктуру, включающую модули разбора заголовков протоколов (полный список требований будет приведен ниже). Построенная по такому принципу система позволит в полной мере использовать преимущества оффлайн-анализа, а также осуществлять разбор заголовков протоколов и восстановление потоков передаваемых данных в режиме онлайн. Основной результат, представляемый в статье – объектная *модель* представления сетевых данных, которая будет использоваться двумя системами.

Требования к системам глубокого анализа пакетов, работающим в онлайн режиме, достаточно проработаны: в 2012 году был опубликован стандарт МСЭ-Т [5]. В свою очередь для аналогичных оффлайн систем требования не уточнялись. В разделе 2 составляется список уточненных функциональных требований для каждой из систем, из которых естественным образом формулируются требования к модели представления данных. В разделе 3 описана модель данных, используемая существующими анализаторами сетевого трафика, такими как Snort [6], Wireshark [7], The Bro Network Security Monitor [8]. Особое внимание уделяется ее недостаткам и ограничениям. Способ преодоления этих недостатков в виде новой модели представления сетевых данных приводится в Разделе 4. В разделе 5 делаются итоговые заключения.

2. Требования к системе разбора

Здесь и далее будем считать, что данные по сети передаются посредством пакетов. Каждый сетевой пакет состоит из управляющей информации и полезной нагрузки. В процессе разбора в пакете выделяются заголовки протоколов, анализируются значения полей этих заголовков. Если структура заголовка определяется спецификацией, то полезная нагрузка может

содержать произвольным образом организованные данные, хотя обычно представляет собой пакет протокола более высокого уровня – анализатор должен самостоятельно определить, какой это протокол. В случае успеха разбор будет продолжен. Рис. 1 иллюстрирует этот процесс.

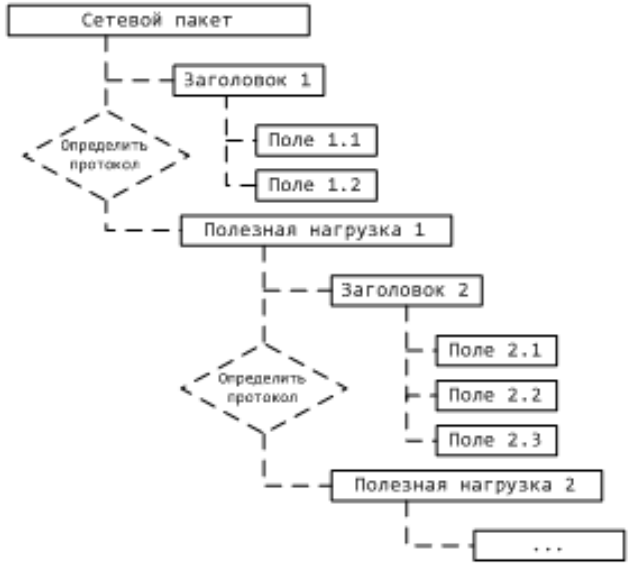

Рис. 1 – Выделение и разбор заголовков протоколов.

В соответствии с моделью OSI содержимое пакета может быть проинтерпретировано посредством стека заголовков сетевых протоколов. Обычно порядок следования этих заголовков естественный – от протоколов более низкого уровня к протоколам более высокого уровня. Однако при организации туннельного соединения порядок следования заголовков может быть нарушен. Система разбора должна корректно обрабатывать подобные ситуации.

Некоторые сетевые протоколы (например, IPv4 [9]) характеризуются максимально допустимым размером полезной нагрузки, передаваемой в одном пакете (Maximum Transmission Unit, или MTU). Соответственно, если величина MTU превышена, полезная нагрузка разбивается на части допустимого размера и передается посредством нескольких сетевых пакетов. В таких случаях система разбора должна проводить дефрагментацию (рис. 2).

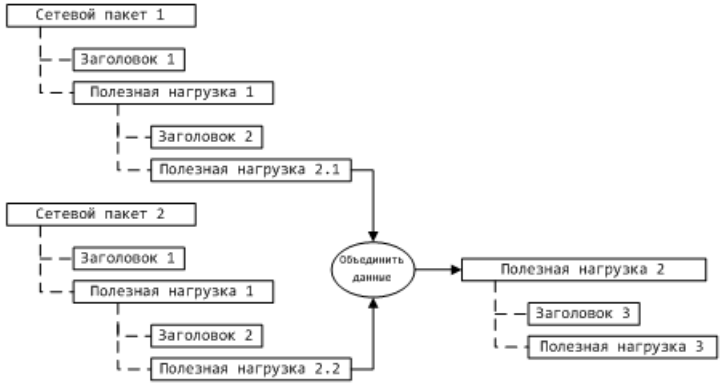

Рис. 2 – Дефрагментация данных (объединение).

Для обеспечения большей безопасности соединений многие протоколы (в частности, TLS [10, 11, 12], SSH [13]) осуществляют передачу данных в зашифрованном виде. Для проведения разбора зашифрованных данных необходимо их предварительно расшифровать, использовав предоставленный пользователем ключ (рис. 3). Обобщая, можно сказать, что система разбора должна предоставлять пользователю интерфейс для добавления недостающей информации, необходимой при проведении разбора. Заметим, что это требование не является обязательным согласно рекомендации МСЭ-Т.

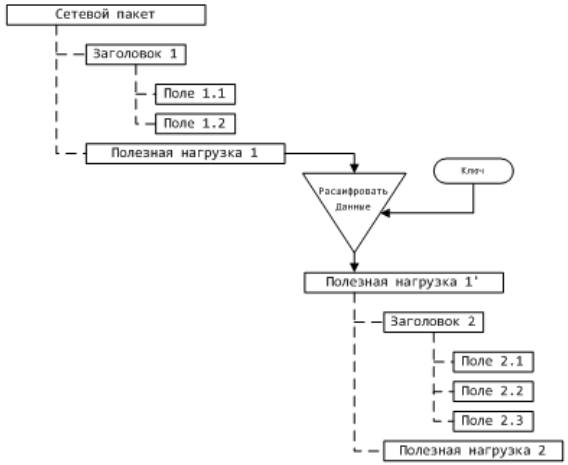

Рис. 3 – Дешифрование данных (преобразование).

При анализе трафика неизбежно возникают *ошибки разбора*. Под ошибкой разбора понимается несоответствие между спецификацией протокола (кодом

разборщика) и сетевым пакетом, разбор которого осуществляется согласно этой спецификации. Причины возникновения ошибок разбора различны:

- ошибки в коде разборщиков,
- недокументированные возможности протоколов,
- искажения данных при передаче по сети.

В системе должна присутствовать возможность локализации данных, разбор которых приводит к возникновению ошибки. При этом, если обнаруженная ошибка не является критической, анализ должен продолжаться.

Как правило, анализаторы сетевого трафика имеют модульную структуру (Wireshark, The Bro Network Security Monitor). Это обусловлено тем, что со временем появляются новые сетевые протоколы, и их необходимо поддерживать. Расширять систему, в которой все разборщики сосредоточены в одном функциональном модуле, затруднительно. В случае модульной архитектуры для каждого протокола создается отдельный модуль, в котором локализована функциональность по работе с этим протоколом. Возникает дополнительный вопрос о зависимостях: при добавлении нового модуля, необходимо "сообщить" о его существовании остальным модулям. Вносить изменения в код существующих модулей крайне неудобно и неэффективно – это потенциальный источник ошибок. К тому же, по окончании правки потребуется повторная сборка модуля. Поэтому необходимо реализовать механизм, позволяющий добавлять новые разборщики (модули разбора) без внесения изменений в уже существующие функции разбора.

Глубокий анализ сетевого трафика предполагает проведение *полного* разбора. Это разбор заголовков протоколов всех уровней в соответствии с моделью OSI, а также сохранение восстановленных потоков передаваемых данных. Возможность проведения полного разбора является ключевым требованием к системе.

С учетом сформулированных требований предлагается следующий (основной) сценарий эксплуатации. Посредством системы онлайн-анализа проводится разбор пакетов (некоторого сетевого интерфейса) в режиме 24/7. *Результаты разбора* сохраняются на жестком диске. Под результатами здесь понимается множество всех восстановленных потоков (см. ниже) всех протоколов. Аналитик с некоторой периодичностью просматривает сообщения об ошибках, выдаваемые системой разбора. Если количество ошибок, возникающих при работе разборщика, превышает предварительно заданное пороговое значение, принимается решение о необходимости доработки кода этого разборщика. Для последующей отладки кода разборщика сохраняется сетевая трасса. Решение о внесении изменений в код принимается по результатам оффлайн-анализа сохраненной трассы.

Фактически работа аналитика с системой оффлайн-анализа заключается в получении и накоплении достаточных знаний о структуре данных для усовершенствования на их основе системы онлайн-анализа. Поэтому крайне важна совместимость модулей разбора заголовков протоколов для двух систем. Обеспечение такой совместимости может быть достигнуто только при достаточном уровне проработки архитектуры этих систем.

Главное, но вполне естественное ограничение системы онлайн-анализа – это ресурсы вычислительной машины. Поскольку поток входных данных потенциально бесконечен, время от времени или по наступлению некоторого события необходимо сохранять результаты разбора на жесткий диск (или передавать их другому анализатору). Также заметим, что система онлайн-анализа не предполагает никакой визуализации структуры разобранных данных.

Система оффлайн-анализа напротив имеет дело с сетевой трассой конечного размера (фактически, отсутствует ограничение, связанное с вычислительными ресурсами). Основное предназначение данной системы – отладка разборщиков. Наглядное представление (визуализация) структуры разобранных пакетов существенно упрощает процесс отладки. Также может оказаться полезным удобный механизм навигации между пакетами и восстановленными потоками.

3. Существующие анализаторы сетевого трафика

Большинство существующих анализаторов поддерживает проведение как онлайн-, так и оффлайн-анализа. В первом случае в качестве источника пакетов выступает сетевое оборудование, во втором – файл сетевой трассы. Разбор пакета заключается в выделении полей всех присутствующих в нем заголовков сетевых протоколов. Под выделением поля подразумевается сопоставление ему некоторого непрерывного *блока* данных известного размера. Как следствие, выделенный блок может приобрести некоторую семантику, например, блок, задающий длину пакета.

Большинство существующих систем разбора оперирует такими понятиями, как пакет, протокол, блок данных. Соответствующая такому подходу модель (далее *базовая модель*) представления данных показана на рис. 4.

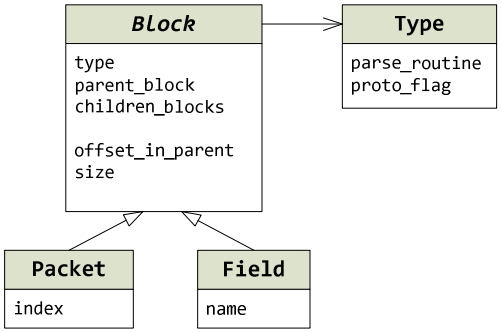

Рис. 4 – Базовая модель представления данных.

Каждый блок хранит указатель на *родительский* блок (parent_block), а также список указателей на *дочерние* блоки (children_blocks). Родительский блок – это блок, внутри которого содержится данный блок. Дочерний блок – это блок, для которого данный блок является родительским. Таким образом поддерживается отношение вложенности блоков, благодаря чему можно ассоциировать сетевой пакет (буфер данных) с *деревом* его разбора. Заметим, что размер блока равен сумме размеров всех его дочерних блоков (если таковые имеются).

Блок обладает *размером* (size) и *смещением* (offset_in_parent) по отношению к родительскому блоку. Блок обладает *типом* (type). Тип определяет разборщик (parse_routine), который будет использоваться при анализе данных соответствующего блока. Дополнительно тип может обладать признаком (proto), указывающим на то, что блок с таким типом представляет собой заголовок сетевого протокола. Этот флаг позволяет группировать блоки и выделять заголовки протоколов при отображении структуры разобранных пакетов.

Каждый блок обладает дополнительным атрибутом: либо *именем* (name – строка), либо порядковым номером (index – целочисленная переменная). Нумерованные блоки описывают сетевые пакеты и элементы массивов, именованные – поля заголовков протоколов.

Обработка данных в соответствии с базовой моделью происходит следующим образом. На вход инструменту подается буфер данных (сетевая трасса в случае оффлайн-анализа). В соответствии с типом данных буфера выбирается нужный разборщик. Разборщик выделяет блоки и при необходимости вызывает другие функции разбора. Результатом работы такой системы является некоторое древовидное представление буфера, в котором каждому выделенному блоку соответствует вершина дерева. Система разбора позволяет аналитику просматривать данные, соответствующие каждому блоку.

Перечислим основные недостатки базовой модели представления данных:

1. Каждый разборщик должен самостоятельно определять, какие функции разбора следует вызывать. Это лишает систему гибкости: при добавлении новых типов потребуется модифицировать функции разбора существующих типов так, чтобы они могли использовать добавленные типы. На практике это означает, что для предоставления возможности добавлять новые функции разбора, разработчик должен открыть исходный код всех реализованных разборщиков. Ни один коммерческий анализатор таких возможностей не предоставляет.
2. Древовидное описание структуры не позволяет описывать пакеты, содержащие зашифрованные или сжатые данные. В то же время объем данных, передаваемых по сети в модифицированном виде, постоянно растет.
3. Описание структуры единственного входного буфера, содержащего все пакеты, не позволяет описывать *сборку сеансов*. Под сеансом в данном случае понимается восстановленная (частично или полностью) единица передачи данных потокового протокола, отправленная (или полученная) посредством более чем одного сетевого пакета. В некоторых инструментах восстановление сеансов реализовано дополнительными модулями (например, Snort, The Bro Network Security Monitor), однако эти сеансы не подвергаются дальнейшему разбору. В то же время, анализ собранных сеансов позволяет восстанавливать высокоуровневые данные прикладных протоколов, тем самым проясняя логику взаимодействия между узлами сети.

Заметим также, что большинство анализаторов предназначено либо для длительной работы в режиме сбора статистики без проведения полного разбора и сохранения структуры данных, либо для кратковременной работы в режиме накопления данных, их разбора и последующего отображения структуры.

4. Формальное описание модели представления данных

Здесь и далее будем называть декларируемую модель *расширенной* по сравнению с базовой моделью.

В расширенной модели процесс разбора заключается в выделении логически непрерывных *блоков* данных. Блок по сути является обобщением понятия поля базовой модели (полем в расширенной модели называется блок, обладающий определенными характеристиками). В процессе анализа блока может потребоваться отдельно обработать какую-либо его часть. Для этого будет создан новый блок, являющийся дочерним по отношению к исходному блоку. Понятия *родительского* и *дочернего* блоков аналогичны понятиям родительского и дочернего полей соответственно. Каждый блок характеризуется положительным числом – размером. В процессе разбора каждому блоку в соответствие ставится *тип*. Тип (разбора) определяет разборщик, посредством которого обрабатываются данные блока. Тип обладает уникальными именем.

Для описания *сборки сеансов* вводится понятие *буфера данных*. Данные блоков могут быть добавлены в конец буфера посредством копирования. Буфер характеризуется размером: в каждый момент времени размер буфера определяется как сумма размеров добавленных в него блоков данных. Содержимое буфера анализируется (посредством разборщиков) так же, как и данные блоков. Здесь возникает необходимость различать блоки, владеющие буфером – *потоки*, и блоки, не владеющие буфером – *фрагменты*. Для случаев, когда требуется сборка сеанса (в частности, при разборе TCP-пакетов [14]), необходимо создать поток, после чего добавить в его буфер нужные данные. Схематично этот процесс показан на рис. 5. Заметим, что поток может быть создан или дополнен любой функцией разбора.

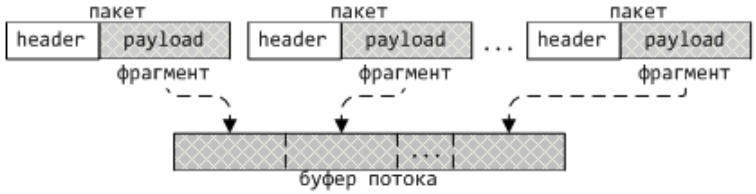

Рис. 5 – Добавление данных нескольких фрагментов в буфер потока.

Также отметим, что данные фрагмента локализуются посредством *смещения* относительно начала данных блока-родителя. Для случаев, когда в качестве родителя выступает поток, смещение задается относительно начала буфера этого потока.

Функциональность по сборке сеансов обеспечивает последовательное повышение уровня (согласно модели OSI) представления данных: по окончанию сборки проводится разбор сеанса, в процессе которого, возможно, будут восстановлены сеансы для протоколов более высокого уровня (рис. 6).

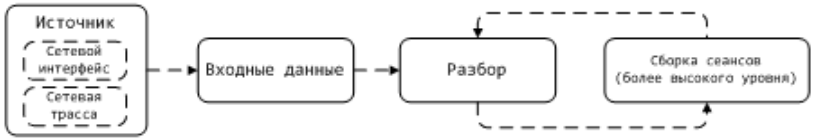

Рис. 6 – Схема работа с данными.

Для того, чтобы система разбора могла работать с модифицированными (например, зашифрованными) данными, вводится понятие *замещающего* фрагмента (фрагменты, не являющиеся замещающими, далее будем называть *простыми*). Проводить разбор зашифрованных данных невозможно – предварительно необходимо провести дешифровку (аналитику могут быть известны алгоритмы дешифрования, а также значения используемых ключей). Дешифрованные данные помещаются в предварительно созданный буфер. Этот буфер ассоциируется с тем же фрагментом, который описывает зашифрованные данные. Таким образом, фрагмент может обладать дополнительным буфером памяти, причем дальнейшему разбору подлежат данные именно этого (*замещающего*) буфера. Предложенный подход естественным образом обобщается для любой модификации данных, допускающей обратное преобразование. Дочерние блоки замещающего фрагмента характеризуются смещением относительно начала данных соответствующего замещающего буфера. Заметим, что размер замещающего буфера в общем случае никак не связан с размером *оригинальных* данных фрагмента. Также отметим, что для описания замещающих фрагментов в рамках древовидной структуры необходимо добавить отличительный признак для вершин, владеющих буфером – будем называть такие вершины *источниками данных*. Тогда простой фрагмент определяется смещением относительно "ближайшего" (при переходе к родительскому блоку) источника данных.

Для реализации в системе универсальных (не требующих внесения изменений в код при добавлении новых типов) разборщиков вводится понятие *распознавателя*. Распознаватель – это функция, которая по данным блока и, возможно, некоторой дополнительной информации определяет тип этого блока. Перед началом разбора блока его тип либо известен, либо нет. В последнем случае необходимо определить тип с помощью распознавателей. Система разбора предоставляет аналитику функции для регистрации распознавателей, а также отвечает за корректное использование уже зарегистрированных распознавателей.

Чаще всего функциональность по распознаванию требуется для полей в заголовках протоколов, допускающих данные различных типов. Например, поле "Payload" в заголовке протокола IPv4 может содержать данные протоколов TCP, UDP [15] и т. д. В данном случае тип определяется значением поля "Protocol" того же заголовка. Более формально, для

определения типа фрагмента используются данные блока-родителя. Такие распознаватели будем называть *распознавателями полей*. При регистрации распознавателя поля в системе необходимо указать имя соответствующего поля, а также тип блока-родителя. Распознаватель поля применяется только к полю с заданным именем при условии, что соответствующий родительский блок обладает заданным типом.

Для определения типа собранных сеансов используются *распознаватели потоков*. Распознаватель потока на вход получает только данные соответствующего буфера. Дополнительно может использоваться *тип сборки* потока (тип сборки потока выставляется в соответствии с типом блоков, данные которых попали в буфер потока). Тип сборки позволяет ограничить (при необходимости) множество применяемых к потоку распознавателей. Такие распознаватели будем называть *распознавателями потоков с известным типом сборки*. Если распознаватель потока не использует тип сборки, то его также можно применять для определения типа фрагмента. Такие распознаватели будем называть *распознавателями потоков с неизвестным типом сборки*.

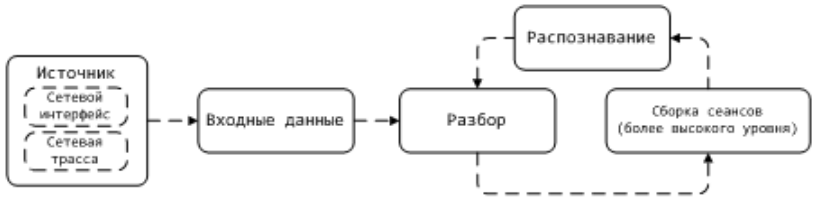

Рис. 7 – Дополненная схема работы с данными.

Распознаватели естественным образом дополняют схему работы с данными в рамках системы (рис. 7). Они служат основным средством обеспечения независимости между функциями (и, как следствие, модулями) разбора: при добавлении в систему новых типов достаточно зарегистрировать соответствующий распознаватель. В противном случае (когда функциональность по распознаванию сосредоточена в функциях разбора) требуется вносить изменения в код разборщиков. Отметим также, что распознаватели могут быть размещены в любом модуле разбора.

Предложенные модификации ликвидируют перечисленные ранее недостатки базовой модели. Однако возникает новая проблема, связанная со сборкой сеансов: при добавлении данных соответствующий сеанс необходимо однозначно идентифицировать. В качестве примера рассмотрим протокол IPv4. Этот протокол используется для передачи данных между узлами сети, которые идентифицируются посредством уникальных IP-адресов. Однако эта уникальность имеет место только внутри определенной IP-подсети. В сетевом трафике, как правило, представлено множество таких подсетей. Кроме того, пакеты одной IP-сети могут быть вложены в пакеты другой IP-сети. Таким

образом, для идентификации IP-сеанса в рамках глобальной сети недостаточно пары IP-адресов. Посредством протокола TCP происходит передача данных между парой портов – отправителя и получателя. То есть для идентификации TCP-сеанса необходимо задать значения портов. Обобщая, можно сказать, что:

- для каждого протокола понятие сеанса различно, то есть имеют место различные *типы сеансов*
- сетевой протокол определяет набор идентификационных характеристик (ключ) для сеансов этого протокола
- уникальность идентификационных характеристик сеанса, определяемых протоколом, имеет место в рамках некоторого *контекста*

Для IP-сеанса таким контекстом является некоторая IP-подсеть, а для TCP-сеанса – IP-сеанс в рамках IP-подсети. Заметим, что чем выше уровень протокола в модели OSI, тем большим набором идентификационных характеристик должен обладать сеанс этого протокола в рамках глобальной сети. Для TCP-сеанса, в частности, этот набор включает характеристики IP-сеанса, а также сам IP-сеанс. Фактически имеет место дерево контекстов, где каждая вершина определяет набор характеристик для соответствующих сеансов, причем этот набор содержит все характеристики родительской вершины и хотя бы одну дополнительную (рис. 8). Для корня дерева набор таких характеристик пуст.

С каждой вершиной дерева контекстов будем ассоциировать набор *экземпляров контекста*. Экземпляр контекста – это контекст, для которого известны значения всех идентификационных характеристик.

Понятие потока, введенное ранее, в полной мере описывает сеанс, причем тип сборки потока (по определению) является типом сеанса. Сборка (и последующее хранение) потока осуществляется в рамках экземпляра контекста.

Для описания контекстов расширим введенное ранее понятие типа, добавив в него признак необходимости создания контекста. Если тип обладает данным признаком, то перед разбором блока соответствующего типа ядро системы осуществит *переключение* контекста (в частности, перед разбором заголовка протокола IPv4). В каждый момент проведения анализа ровно один контекст является *активным*. Переключение заключается в смене активного контекста – если нужного контекста не существует к моменту переключения, то он будет создан. Будем называть *типом контекста* тип блока, разбор которого потребовал создания этого контекста.

Менее формально, контексты и экземпляры контекстов предназначены для группировки блоков одного уровня вложенности. Фактически речь идет о классификации трафика (подробнее о методах классификации в статье [16]). Критерий группировки определяется типом (контекста) и задается функцией, которая на основе анализа содержимого блока относит его к конкретной

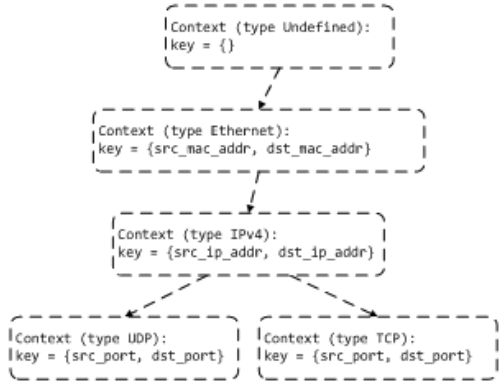

Рис. 8 – Пример дерева контекстов.

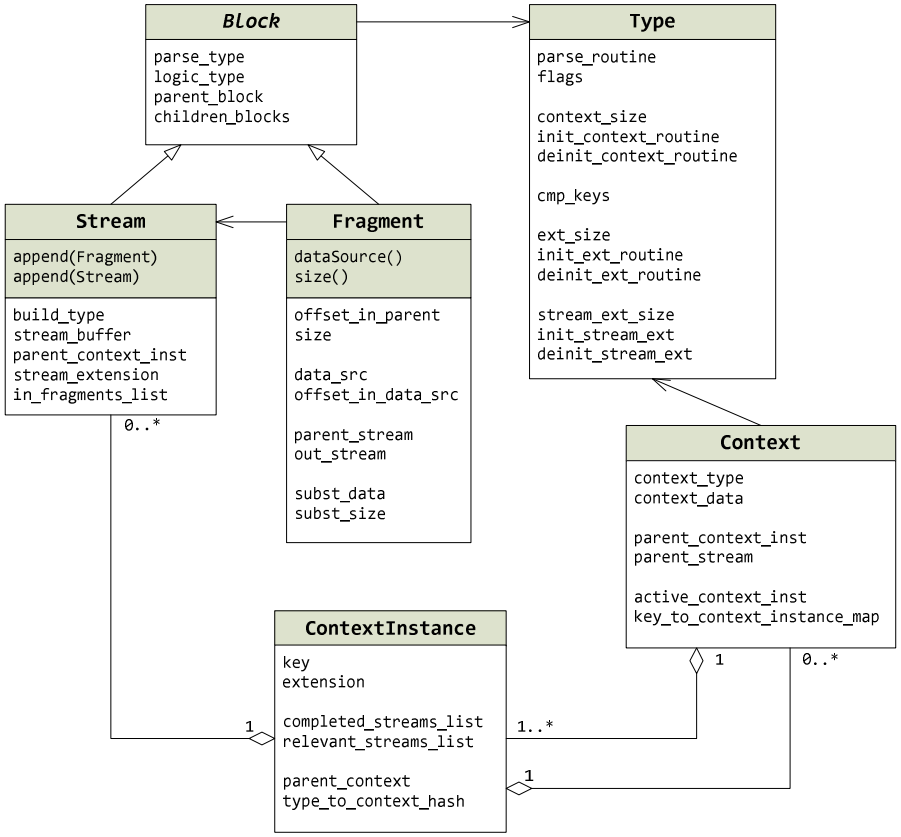

Рис. 9 – Расширенная модель представления данных.

группе (каждый экземпляр в рамках некоторого контекста описывает одну такую группу).

Сборка потоков проводится в рамках одного экземпляра контекста. Это значит, что блоки, принадлежащие разным экземплярам контекста, не могут стать частями одного и того же потока.

Объектная модель, учитывающая все перечисленные особенности, представлена на рис. 9. Подчеркнем, что на уровне блоков поддерживается семантика (атрибут logic_type): в соответствии с типом (атрибут flags) блок является либо *структурой* (по аналогии с типом "struct" языка Си), либо *последовательностью*. Соответственно, дочерние блоки в первом случае называются *полями*, а во втором – *элементами последовательности*. Каждое поле характеризуется именем, каждый элемент последовательности – порядковым номером. Расширенная модель, таким образом, полностью покрывает функционал базовой модели.

Также добавим, что в рамках экземпляра контекста потоки идентифицируются посредством ключа (атрибут stream_extension), размер (атрибут stream_ext_size) и структура которого определяются типом соответствующего контекста.

5. Заключение

В статье предложена объектная модель представления данных, применяемая в рамках инструментов оффлайн- и онлайн-анализа. В отличие от базовой модели, лежащей в основе большинства существующих инструментов анализа сетевого трафика, расширенная модель предоставляет функциональность по сборке сеансов, поддерживает работу с модифицированными данными, позволяет вести разработку модулей разбора независимо. Схожую модель предлагает инструмент Wireshark, однако она обладает существенным недостатком: сборка сеансов целиком ложится на плечи разработчика модулей разбора. На уровне ядра Wireshark отсутствуют аналоги таких понятий как поток, контекст и экземпляр контекста: для каждого модуля (при необходимости проведения сборки) их нужно создавать отдельно. В результате, код разборщиков оказывается перегруженным логикой сборки. С другой стороны, предложенная в статье модель вводит абстракции для сборки сеансов на уровне ядра, что позволяет существенно упростить (и сократить) код разборщиков и избежать дополнительных ошибок.

Список литературы

[1]. P. Tsankov, M. T. Dashti, D. Basin. SECFUZZ: Fuzz-testing Security Protocols // Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 1-7, 2012

[2]. Н. В. Пакулин, В. З. Шнитман, А. В. Никешин. Автоматизация тестирования соответствия для телекоммуникационных протоколов // Труды Института системного программирования РАН, том 26, выпуск 1, 2014, стр. 109-148. DOI: 10.15514/ISPRAS-2014-26(1)-4

[3]. Karen Scarfone, Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS) // National Institute of Standards and Technology Special Publication 800-94, 127 pages, February 2007

[4]. Ю. В. Маркин, А. С. Санаров. Обзор современных инструментов анализа сетевого трафика // Препринты ИСП РАН, №27, 2014

[5]. Рекомендация МСЭ-Т Y.2770, Требования к углубленной проверке пакетов в сетях последующих поколений, издание 1.0, 20.11.2012

[6]. Snort. http://www.snort.org/, дата обращения 07.10.2015

[7]. Wireshark. http://www.wireshark.org/, дата обращения 07.10.2015

[8]. The Bro Network Security Monitor. http://www.bro.org/, дата обращения 07.10.2015

[9]. IETF RFC 791. Information Sciences Institute, Internet Protocol, September 1981

[10]. IETF RFC 5246. T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

[11]. Н. В. Пакулин, В. З. Шнитман, А. В. Никешин. Разработка тестового набора для верификации реализаций протокола безопасности TLS // Труды Института системного программирования РАН, том 23, 2012, стр. 387-404. DOI: 10.15514/ISPRAS-2012-23-22

[12]. А. В. Никешин, Н. В. Пакулин, В. З. Шнитман. Тестирование реализаций клиента протокола TLS // Труды Института системного программирования РАН, том 27, выпуск 2, 2015, стр. 145-160. DOI: 10.15514/ISPRAS-2015-27(2)-9

[13]. IETF RFC 4251. T. Ylonen, C. Lonvick, The Secure Shell (SSH) Protocol Architecture, January 2006

[14]. IETF RFC 791. Information Sciences Institute, Transmission Control Protocol, September 1981

[15]. IETF RFC 768. J. Postel, User Datagram Protocol, August 1980

[16]. F. Risso, A. Baldini, M. Baldi, P. Monclus, O. Morandi, Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation // IEEE International Conference on Communications (ICC 2008), Beijing (China), pp. 5869-5875, May 2008

Model of Data Handling for In-Depth Analysis of Network Traffic★

[1]A. I. Get'man <thorin@ispras.ru>
[1, 2, 3, 4]V.P. Ivannikov <ivan@ispras.ru>
[1]Yu. V. Markin <ustas@ispras.ru>
[1, 2]V. A. Padaryan <vartan@ispras.ru>
[1]A. Yu. Tikhonov <fireboo@ispras.ru>
[1] Institute for System Programming of the Russian Academy of Sciences, 25, Alexander Solzhenitsyn st., 109004, Moscow, Russia
[2]Lomonosov Moscow State University, 2nd Education Building, Faculty CMC, GSP-1, Leninskie Gory, Moscow, 119991, Russian Federation
[3]Moscow Institute of Physics and Technology, 9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
[4]Higher School of Economics, National Research University, 20 Myasnitskaya Ulitsa, Moscow 101000, Russia

Abstract. The article suggests a new object model of data for in-depth analysis of network traffic. In contrast to the model used by most existing network analyzers, such as Wireshark or Snort, the core of our model supports data streams reassembling and next processing of them. Analysis continues even in case of loss of individual packets. The model supports both stateless and statefull network protocols. State of protocol machine may be stored in a special memory location related to each connection of relevant type. The article stated the requirements for network traffic analysis tools. A high speed data processing in resource-limited environment is the main requirement for online systems. Offline analyzer operates with a network trace of the fixed size, so the processing speed is not so important. It becomes possible to visualize the data structure disassembled. Offline analyzer also traces how network streams formed from packets. The model provides an interface for parsers implemented in the form of dynamic link libraries. It also provides a convenient universal mechanism for binding parsers so one can develop parsers independently. This is achieved through the use of special functions (recognizers) allowing for the data itself to determine which parser should be used. It is crucial for parsers to be compatible with both online and offline analyzers. Our model also provides processing of modified, e.g. compressed or encrypted, data. Unlike Snort the model supports nested tunneling protocols. Actually it forms the basis of the infrastructure for in-depth analysis of network traffic.

Keywords: network traffic analysis; data stream assembling; model of data; recognition of data.

DOI: 10.15514/ISPRAS-2015-27(4)-1

★ This work is supported by RFBR grant 15-07-07652 A

For citation: Get'man A.I., Ivannikov V.P., Markin Yu.V., Padaryan V.A., Tikhonov A.Yu. Model of Data Handling for In-Depth Analysis of Network Traffic. *Trudy ISP RAN/Proc. ISP RAS*, vol. 27, issue 4, 2015, pp. 5-22 (in Russian). DOI: 10.15514/ISPRAS-2015-27(4)-1.

References

[1]. P. Tsankov, M. T. Dashti, D. Basin. SECFUZZ: Fuzz-testing Security Protocols // Proceedings of the 7th International Workshop on Automation of Software Test (AST 2012), pp. 1-7, 2012

[2]. A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. Avtomatizatsiya testirovaniya sootvetstviya dlya telekommunikatsionnykh protokolov [Automation of conformance testing for communication protocols] // Trudy ISP RAN [The Proceedings of ISP RAS], 2014, vol. 26, no. 1, pp. 109-148 (in Russian). DOI: 10.15514/ISPRAS-2014-26(1)-4

[3]. Karen Scarfone, Peter Mell. Guide to Intrusion Detection and Prevention Systems (IDPS) // National Institute of Standards and Technology Special Publication 800-94, 127 pages, February 2007

[4]. Yu. V. Markin, A. S. Sanarov. Obzor sovremennykh instrumentov analiza setevogo trafika [The modern network traffic analyzers overview] // Preprinty ISP RAN [Preprints of ISP RAS], №27, 2014 (in Russian)

[5]. Recommendation ITU-T Y.2770, Requirements for deep packet inspection in next generation networks, edition 1.0, 2012.11.20

[6]. Snort. http://www.snort.org/, access date: 2015.10.07

[7]. Wireshark. http://www.wireshark.org/, access date: 2015.10.07

[8]. The Bro Network Security Monitor. http://www.bro.org/, access date: 2015.10.07

[9]. IETF RFC 791. Information Sciences Institute, Internet Protocol, September 1981

[10]. IETF RFC 5246. T. Dierks, E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

[11]. A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. Razrabotka testovogo nabora dlya verifikatsii realizatsii protokola bezopasnosti TLS [Creation of a test suite for verification of the TLS security protocol] // Trudy ISP RAN [The Proceedings of ISP RAS], 2012, vol. 23, pp. 387-404 (in Russian). DOI: 10.15514/ISPRAS-2012-23-22

[12]. A. V. Nikeshin, N. V. Pakulin, V. Z. Shnitman. Testirovanie realizatsii klienta protokola TLS [TLS clients testing] // Trudy ISP RAN [The Proceedings of ISP RAS], 2015, vol. 27, no. 2, pp. 145-160 (in Russian). DOI: 10.15514/ISPRAS-2015-27(2)-9

[13]. IETF RFC 4251. T. Ylonen, C. Lonvick, The Secure Shell (SSH) Protocol Architecture, January 2006

[14]. IETF RFC 791. Information Sciences Institute, Transmission Control Protocol, September 1981

[15]. IETF RFC 768. J. Postel, User Datagram Protocol, August 1980

[16]. F. Risso, A. Baldini, M. Baldi, P. Monclus, O. Morandi, Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation // IEEE International Conference on Communications (ICC 2008), Beijing (China), pp. 5869-5875, May 2008